实验八　状态机及键盘输入

2020 年秋季学期

杨飞洋 191220138

目录

1. 实验目的：

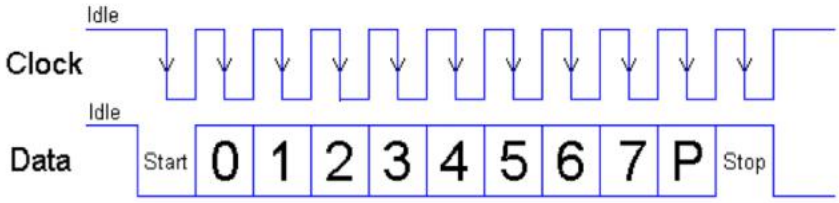
学习状态机的工作原理，了解状态机的编码方式，并利用 PS/2 键盘输入实现简单状态机的设计。

2. 实验原理（知识背景）

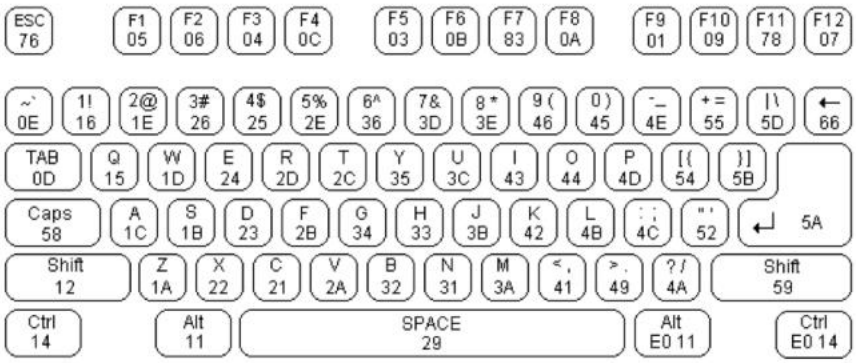
有限状态机 FSM（Finite State Machine）简称状态机，是一个在有限个状态间进行转换和动作的计算模型。有限状态机含有一个起始状态、一个输入列表（列表中包含了所有可能的输入信号序列）、一个状态转移函数和一个输出端，状态机在工作时由状态转移函数根据当前状态和输入信号确定下一个状态和输出。状态机一般都从起始状态开始，根据输入信号由状态转移函数决定状态机的下一个状态。

PS/2 是个人计算机串行 I/O 接口的一种标准，因其首次在 IBM PS/2（Personal System/2）机器上使用而得名，PS/2 接口可以连接 PS/2 键盘和 PS/2 鼠标。所谓串行接口是指信息是在单根信号线上按序一位一位发送的。

当用户按键或松开时，键盘以每帧 11 位的格式串行传送数据给主机，同时在 PS2_CLK 时钟信号上传输对应的时钟（一般为 10.0–16.7kHz）。第一位是开始位（逻辑 0），后面跟 8 位数据位（低位在前），一个奇偶校验位（奇校验）和一位停止位（逻辑 1）。每位都在时钟的下降沿有效下图显示了键盘传送一字节数据的时序。



键盘扫描码：



断码为 F0h

3. 实验环境/器材

系统：windows10
开发软件：Quartus 17.1 Lite
开发板：DE10 Standard
仿真环境：ModelSim
芯片：Cyclone V ， 5CSXFC6D6
键盘：PS2 键盘

4. 设计思路

一开始想直接用老师提供的模块，应该是因为对代码理解不深的缘故，总是有问题。最后不得不自己重新写了一个简易版的接受键盘数据的模块。
输入输出：

```
module ps2_keyboard1(
    input ps2_clk,
    input ps2_data,
    output reg pressed = 0,
    output reg [7:0] data = 0,
    output reg [7:0] ascii = 0,
    output reg [7:0] times = 0,
    output reg shift = 0,
    output reg ctrl = 0,
    output reg caps = 0
);
```

ps2_clk 接受键盘时钟，
ps2_data 接受键盘输送过来的数据，
pressed 显示当前键盘有无按键按下，
data 输出键盘按键的扫描码，
ascii 输出键盘按键对应的 ascii 码，
times 记录按键的次数，
shift、ctrl 显示当前 shift 键和 ctrl 键是否被按下，
caps 显示 caps 键的状态。

引进几个变量：

```
reg [10:0] buffer;
reg [3:0] pointer = 0;
reg break = 0;
```

buffer 记录键盘的 11 位数据，pointer 相当于一个指针，将 ps2_data 输送到对应的 buffer 位置，break 记录是否松开按键。
在开始之前还有两个准备工作要做，一是写一个 7 段数码管的显示模块，二是初始化一个.mif 文件用来对应扫描码和 ascii 码

7 段数码管显示模块：

```
module trsltr(number,s);
    input [3:0]number;
    output reg [6:0]s;

    always @(*)
        case(number)
            0:s = 7'b1000000;
            1:s = 7'b1111001;
            2:s = 7'b0100100;
            3:s = 7'b0110000;
            4:s = 7'b0011001;
            5:s = 7'b0010010;
            6:s = 7'b0000010;
            7:s = 7'b1111000;
            8:s = 7'b0000000;
            9:s = 7'b0010000;
            10:s = 7'b0001000;
            11:s = 7'b0000011;
            12:s = 7'b1000110;
            13:s = 7'b0100001;
            14:s = 7'b0000110;
            15:s = 7'b0001110;
            default:s = 7'b1111111;
        endcase

endmodule
```

两个.mif 文件（因为 shift 会改变键值）
不按 shift 的 lower.mif:

Addı	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	0	0	0	0	0	0	0	0	.....
8	0	0	0	0	0	0	96	0	..... `.
16	0	0	0	0	0	113	49	0	.....q1.
24	0	0	122	115	97	119	50	0	..zsaw2.
32	0	99	120	100	101	52	51	0	.cxde43.
40	0	32	118	102	116	114	53	0	. vftr5.
48	0	110	98	104	103	121	54	0	.nbhgy6.
56	0	0	109	106	117	55	56	0	..mju78.
64	0	44	107	105	111	48	57	0	.,kio09.
72	0	46	47	108	59	112	45	0	../l;p-.
80	0	0	39	0	91	61	0	0	..'.[=..
88	0	0	13	93	0	92	0	0	...].\..
96	0	0	0	0	0	0	8	0	.....
104	0	0	0	0	0	0	0	0	.....

按下 shift 的 upper.mif:

Addı	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	0	0	0	0	0	0	0	0	.....
8	0	0	0	0	0	0	126	0	.....~.
16	0	0	0	0	0	81	33	0	.....Q!.
24	0	0	90	83	65	87	64	0	..ZSAW@.
32	0	67	88	68	69	36	35	0	.CXDE$#.
40	0	0	86	70	84	82	37	0	..VFTR%.
48	0	78	66	72	71	89	94	0	.NBHGY^.
56	0	0	77	74	85	38	42	0	..MJU&*.
64	0	60	75	73	79	41	40	0	.<KIO)(.
72	0	62	63	76	58	80	95	0	.>?L:P_.
80	0	0	34	0	123	43	0	0	..".{+..
88	0	0	0	125	0	124	0	0	...}.|..
96	0	0	0	0	0	0	0	0	.....

接下来写接受键盘数据的逻辑。
首先将扫描码和 ascii 码的对应写进数组：

```
(* ram_init_file = "lower.mif" *) reg [7:0] lower[255:0];
(* ram_init_file = "upper.mif" *) reg [7:0] upper[255:0];
```

即 lower[扫描码] = ascii 码(或 upper[扫描码] = ascii 码)

首先将收到的数据读入 buffer，然后 pointer 自增，到 10 变为 0.

```
always @(negedge ps2_clk) begin
    buffer[pointer] = ps2_data;
    pointer = (pointer+1) % 11;
```

如果 pointer == 0，代表读完一组数据，将 buffer 中的数据输出到 data

```
if(pointer == 0) begin
    data = buffer[8:1];
    if(break) begin
        break = 0;
        if(data == 8'h12) shift = 0;
        if(data == 8'h14) ctrl = 0;
        pressed = 0;
        data = 0;
        ascii = 0;
    end
```

break 是之后记录的数据，如果读到 F0，就表示按键松开，除了 caps 不受影响，其他值都要置零。

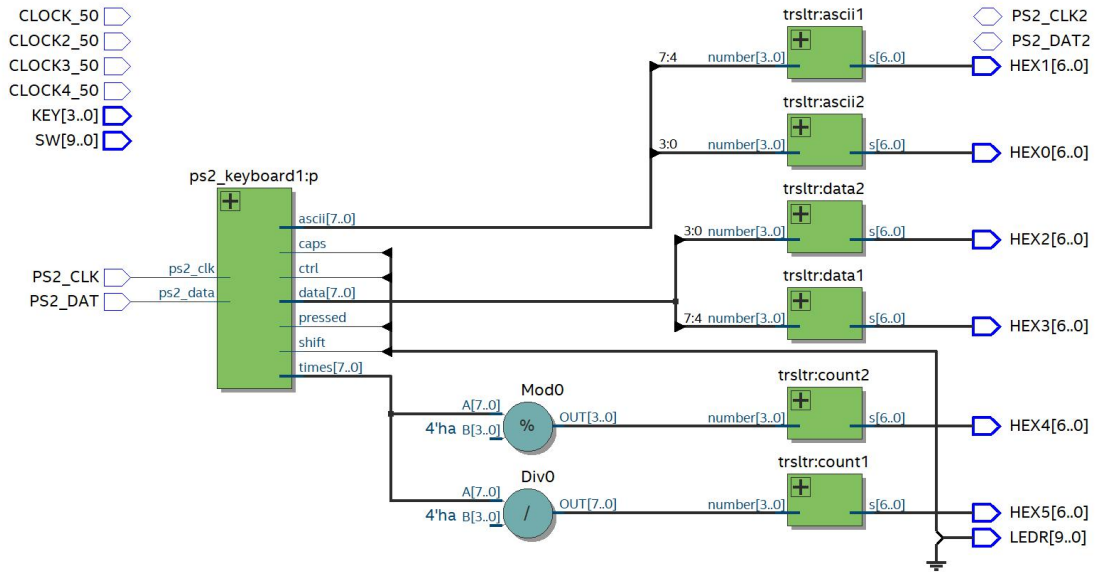
如果之前没有 break 信号，即之前没有收到 F0 的扫描码，则处理 data，由于 shift、caps、ctrl 都是功能按键，所以我就没有在按下 shift、caps、ctrl 的时候 times++。又因为 shift 和 caps 全 0 或全 1 对键值没有影响，但只有一个 1 时是有影响的，所以用 shift^caps 来指示需不需要用到 upper 数组，如果 data == F0，表示按键松开，break 置为 1.

```
else if(!pressed || shift || ctrl) begin
    case(data)
        8'h12: begin shift = 1;pressed = 1; end
        8'h14: begin ctrl = 1;pressed = 1;end
        8'h58: begin caps = ~caps;pressed = 1;end
        8'hf0: ;
        default: begin
            times = times + 1;
            if(shift ^ caps) //upper
                ascii = upper[data];
            else
                ascii = lower[data];
            pressed = 1;
        end
    endcase
end
if(data == 8'hf0) break = 1;
```

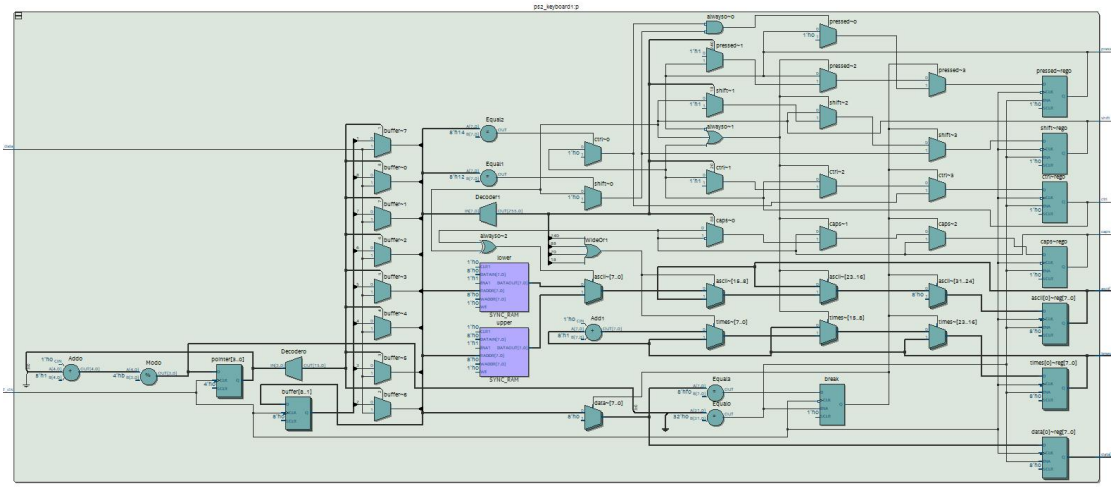
在总模块中的调用：

```
wire [7:0]data;
wire [7:0]ascii;
wire [7:0]times;

ps2_keyboard1 p(PS2_CLK,PS2_DAT,LEDR[0],data,ascii,times,LEDR[1],LEDR[2],LEDR[3]);

trsltr count1(times/10,HEX5[6:0]);
trsltr count2(times%10,HEX4[6:0]);

trsltr data1(data/16,HEX3[6:0]);
trsltr data2(data%16,HEX2[6:0]);

trsltr ascii1(ascii/16,HEX1[6:0]);
trsltr ascii2(ascii%16,HEX0[6:0]);
```

RTL viewer：



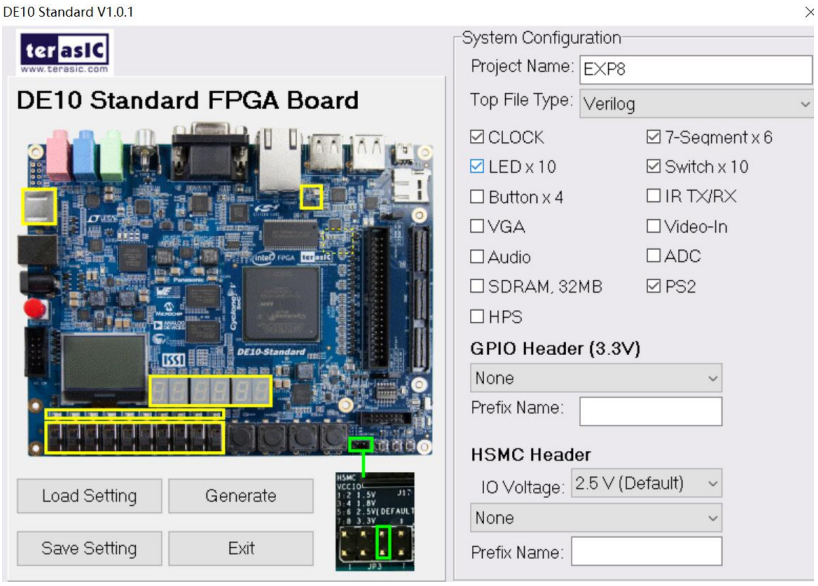
数据接收模块的 RTL viewer：



5.实验步骤

一．利用 systembuilder 建立工程

选择 CLOCK、SWITCH、LEDR、PS2



二．编写 Verilog 代码

写一下相应模块的代码，在顶层文件中做相应的调用，进行编译调试，直至成功。

三．在板子和键盘上进行测试

6.测试方法

在键盘上输入尽可能多的情况，尤其是验证 shift 按键和 caps 按键的功能，同时关注按下的键数是否自增。

7.实验结果

可见同级目录下的 result.mp4

8.实验中遇到的问题

1.第一次用键盘验证时，接受的数据要么是 00 要么是 FF，然后换了块键盘就正常了。
2.一开始没有理清键盘时钟和自身时钟之间的关系，后来统一为键盘时钟，并且是下降沿有效，这个最初也没有注意到。
3.最初没想到用.mif 文件来存储信息，本想着用一堆 case 语句来写，后来想到.mif 文件就好很多了。

9.实验得到的启示

1．一定要学会用分模块编程，会让结构更加清晰，debug 更加容易
2．有问题可以向老师助教请教、和同学讨论、上网搜，很多时候是思路或结构出现了问题。
3. 有时可能真的是设备的问题。

10.意见和建议

1.这种实验测试文件不好测试，希望老师可以直接说清楚不用测试文件，只需要在板子上测试。
2.希望给一些实验原理的指导，不然自己搞有点懵。
3.感觉这样每次的实验都是接触的全新的东西，又无法深究，这样效率有点低。还是希望深究一点东西。